Crud base

```
import mysql.connector
from backend.database import ConexionDB

class BaseCRUD:
    def __init__(self):
        self.db = ConexionDB()

    def _get_connection(self):
        conn = self.db.conectar()
        if conn is None:
            raise RuntimeError("No se pudo conectar a la base de datos")
        return conn

    def _consultar(self, query, params=None, fetch_one=False):
        """
         Notas importantes:
        - cursor(dictionary=True) devuelve dicts (ideal para FastAPI + response_model)
        - fetch_one: devuelve 1 dict o None
        - fetch_all: devuelve lista de dicts ([])
        """
        with self._get_connection() as conn:
            cursor = conn.cursor(dictionary=True)
            cursor.execute(query, params or ())
            if fetch_one:
                return cursor.fetchone()
            return cursor.fetchall()

    def _ejecutar(self, query, params=None):
        """
         Notas importantes:
        - Devuelve lastrowid para INSERT
        - Devuelve rowcount útil para UPDATE/DELETE
        """
        with self._get_connection() as conn:
            cursor = conn.cursor()
            cursor.execute(query, params or ())
            conn.commit()
            return cursor.lastrowid, cursor.rowcount
```

Crud usuarios

```
from backend.crud.base import BaseCRUD
```

```
class UsuariosCRUD(BaseCRUD):
```

1

```
    # ============================
    # CREAR
    # ============================
    def crear(self, usuario):
        """
         Notas importantes:
        - usuario es un modelo Pydantic (UsuarioCreate)
        - Por eso se accede con atributos:
```

```
    # ====================
    # OBTENER TODOS
    # ====================
    def obtener_todos(self):
        sql = """
        SELECT
            id_usuario,
            nombre,
            apellido,
            profesion_oficio,
            empresa,
            telefono,
            telefono_alt,
            email,
            cuil_cuit,
            created_at,
            updated_at
        FROM usuario
        ORDER BY id_usuario DESC
        """
        return self._consultar(sql)

    # ====================
    # OBTENER POR ID
    # ====================
    def obtener_por_id(self, id_usuario
        sql = """
        SELECT
            id_usuario,
            nombre,
            apellido,
            profesion_oficio,
            empresa,
            telefono,
            telefono_alt,
            email,
            cuil_cuit,
            created_at,
            updated_at
        FROM usuario
        WHERE id_usuario = %s
        """
        return self._consultar(sql, (id_usuario,),
Lo bajo
fetch_one=True)
```

```
    # ==========================
    # ACTUALIZAR
    # ==========================
    def actualizar(self, id_usuario: int, usuario_update):
```

```
        """
        □ Notas importantes:
        - usuario_update es un Pydantic model (UsuarioUpdate), NO dict.
        - Convertimos a dict SOLO con campos que vinieron en el body.
        """
        data = usuario_update.model_dump(exclude_unset=True)

        if not data:
            # no vino ningún campo para actualizar
            return None

        # Construimos SQL dinámico: solo actualiza campos enviados
        set_parts = []
        params = []

        for campo, valor in data.items():
            set_parts.append(f"{campo} = %s")
            # EmailStr a str
            if campo == "email" and valor is not None:
                valor = str(valor)
            params.append(valor)

        sql = f"""
        UPDATE usuario SET
            {", ".join(set_parts)}
        WHERE id_usuario = %s
        """
        params.append(id_usuario)

        _, rowcount = self._ejecutar(sql, tuple(params))
        if rowcount == 0:
            return None

        return self.obtener_por_id(id_usuario)

    # ============================
    # ELIMINAR
    # ============================
    def eliminar(self, id_usuario: int):
        sql = "DELETE FROM usuario WHERE id_usuario = %s"
        _, rowcount = self._ejecutar(sql, (id_usuario,))
        return rowcount > 0
```

Crud Notas de pie

```
from backend.crud.base import Base(

class PiesPresupuestoCRUD(BaseCRU
```

```
 #
==============================
======
    # CREAR
    #
==============================
===
    def crear(self, pie_create):
        sql = """
        INSERT INTO pies_presupuesto (
            id_presupuesto,
            titulo,
            texto
```

```
    # ====================
    # OBTENER TODOS
    # ====================
    def obtener_todos(self):
        sql = """
        SELECT
            id_pie,
            id_presupuesto,
            titulo,
            texto,
            created_at,
            updated_at
        FROM pies_presupuesto
        ORDER BY id_pie DESC
        """
        return self._consultar(sql)

    # ====================
    # OBTENER POR ID
    # ====================
    def obtener_por_id(self, id_pie: int)
        sql = """
        SELECT
            id_pie,
            id_presupuesto,
            titulo,
            texto,
            created_at,
            updated_at
        FROM pies_presupuesto
        WHERE id_pie = %s
        """
        return self._consultar(sql, (id_pie
bajo
fetch_one=True)
```

```
    #
==================3=
======
    # ELIMINAR
    #
====================
=====
    def eliminar(self, id_pie: int):
        sql = "DELETE FROM
pies_presupuesto
Abajo
WHERE id_pie = %s"
```